The stubborn problem is stubborn no more

(a polynomial algorithm for 3–compatible colouring and the stubborn list partition problem)

Marek Cygan Marcin Pilipczuk Michał Pilipczuk Jakub Onufry Wojtaszczyk*

Abstract

One of the driving problems in the CSP area is the Dichotomy Conjecture, formulated in 1993 by Feder and Vardi [STOC'93], stating that for any fixed relational structure Γ the Constraint Satisfaction Problem CSP(Γ) is either NP–complete or polynomial time solvable. A large amount of research has gone into checking various specific cases of this conjecture. One such variant which attracted a lot of attention in the recent years is the LIST MATRIX PARTITION problem. In 2004 Cameron et al. [SODA'04] classified almost all LIST MATRIX PARTITION variants for matrices of size at most four. The only case which resisted the classification became known as the STUBBORN PROBLEM. In this paper we show a result which enables us to finish the classification — thus solving a problem which resisted attacks for the last six years.

Our approach is based on a combinatorial problem known to be at least as hard as the STUBBORN PROBLEM — the 3-COMPATIBLE COLOURING problem. In this problem we are given a complete graph with each edge assigned one of 3 possible colours and we want to assign one of those 3 colours to each vertex in such a way that no edge has the same colour as both of its endpoints. The tractability of the 3-COMPATIBLE COLOURING problem has been open for several years and the best known algorithm prior to this paper is due to Feder et al. [SODA'05] — a quasipolynomial algorithm with a $n^{O(\log n/\log\log n)}$ time complexity. In this paper we present a polynomial–time algorithm for the 3-COMPATIBLE COLOURING problem and consequently we prove a dichotomy for the k-COMPATIBLE COLOURING problem.

*Dept. of Mathematics, Computer Science and Mechanics, University of Warsaw, Poland, [cygan@,malcin@,michal.pilipczuk@students.,onufry@]mimuw.edu.pl

1 Introduction

In this paper we consider a variant of the graph colouring problem, namely the k-COMPATIBLE COLOURING problem. We are given a complete graph with each edge assigned one of k possible colours and we want to assign one of those k colours to each vertex in such a way that no edge has the same colour as both of its endpoints. Formally:

k-COMPATIBLE COLOURING (sometimes called EDGE FREE k-COLOURING)
Input: A complete undirected graph $G = (V, E)$ and a function $\mathcal{C} : E \to \{0, \ldots, k-1\}$
Question: Does there exist a function $\phi : V \to \{0, \ldots, k-1\}$ such that for each edge $uv \in E$ either $\phi(u) \neq \mathcal{C}(uv)$ or $\phi(v) \neq \mathcal{C}(uv)$

For $k = 1$ this problem is meaningless, but for $k = 2$ it can be interpreted as a split graphs recognition problem. Indeed, if we consider a graph $G' = (V, \mathcal{C}^{-1}\{1\})$ (i.e., we take only those edges from $e \in E$ for which $\mathcal{C}(e) = 1$) our task is equivalent to partitioning the graph G' into a clique and an independent set. Graphs that can be partitioned in this way are called *split graphs* and can be recognized in linear time [14].

It is known [16] that for $k \geq 4$ the k-COMPATIBLE COLOURING problem becomes NP-complete. However, for $k = 3$ the problem of its tractability has been open for several years. In this paper we show a polynomial–time algorithm for this case.

To compare, the classical colouring problem is NP-complete for $k \geq 3$ and polynomial time solvable for $k \leq 2$. Until now it was not known whether k-COMPATIBLE COLOURING admits such a dichotomy since the previously best algorithm (by Feder et al. from 2005 [10]) has a $n^{O(\log n / \log\log n)}$ time complexity which is an improvement over the $n^{O(\log n)}$ time complexity of an algorithm by Feder and Hell [8].

Related work and motivation We briefly sketch the CONSTRAINT SATISFACTION PROBLEM (CSP) definition in the notation proposed by Feder and Vardi [13]. For a fixed relational structure Γ in the problem CSP(Γ) we are given a second relational structure G and we are asked whether there exists a homomorphism of G to Γ (a mapping $f : V(G) \to V(\Gamma)$ which preserves all the relations). Feder and Vardi [13] in 1993 formulated the following conjecture which remains open and motivates a lot of research in this area.

Conjecture 1 (The Dichotomy Conjecture [13])**.** *For any fixed relational structure* Γ *the problem* CSP(Γ) *is either NP-complete or polynomial time solvable.*

Since then dozens of papers have been written proving this conjecture in several special cases (for a survey see [16]). In particular, Conjecture 1 holds for every relational structure of size two [18] and three [2].

The k-COMPATIBLE COLOURING problem is a variant of full-CSP problems introduced by Feder and Hell in [8], whereas the exact name k-COMPATIBLE COLOURING to the best of our knowledge comes from [15]. Intuitively, in full-CSP problems we restrict ourselves to structures G in which every tuple of elements is restricted by some constraint. A similar variant of CSP studied in the literature is called the LIST MATRIX PARTITION where Γ is represented by an $r \times r$ symmetrical matrix M with entries being subsets of $\{0, \ldots, q-1\}$ for some integer q. We are given a complete graph G with vertices equipped with subsets of $\{0, \ldots, r-1\}$ and edges assigned values from $\{0, \ldots, q-1\}$. We ask whether there exists a function $\phi : V(G) \to \{0, \ldots, r-1\}$ such that for each $v \neq w \in V(G)$, $\phi(v)$ belongs to the set tied to the vertex v and the value associated with the edge vw belongs to the set in the $\phi(v)$-th row and $\phi(w)$-th column of M. A formal description can be found in [9]. It is known that for fixed q, r the LIST MATRIX PARTITION problem enjoys a quasi-dichotomy.

Theorem 2 (Quasi-dichotomy Theorem [7])**.** *For each pair of positive integers r, q and for each symmetrical $r \times r$ matrix M whose entries are subsets of the set $\{0, \ldots, q-1\}$ the* LIST MATRIX PARTITION *problem is either NP-complete or solvable in quasipolynomial time.*

The currently best bound for the quasipolynomial from Theorem 2 due to Feder and Hell [7] is $n^{O(\log n)}$, where $n = |V(G)|$. In order to check whether this quasi-dichotomy is a classical dichotomy several special cases for small values of q and r were studied. In particular, Cameron et al. [3] were able to classify almost all matrices with $r \leq 4$ and $q = 2$. For all classified matrices either a polynomial time algorithm or a NP-completeness proof was given. Interestingly enough, the classified cases were equivalent to numerous classical graph problems such as: 3-colourability, clique cutset, stable cutset, skew partition and split graphs recognition. To underline the significance of the LIST MATRIX PARTITION we recall (as stated in [3]) that the resolution of the *Strong Perfect Graph Conjecture* by Chudnovsky et al. [4] relies in part on decompositions that can be formulated as LIST MATRIX PARTITION instances. The only two matrices that Cameron et al. could not classify are polynomially equivalent to the following problem which came to be called the STUBBORN PROBLEM.

STUBBORN PROBLEM
Input: An undirected graph $G = (V, E)$ and a constraint function $\mathcal{L} : V \to \mathcal{P}(\{1, 2, 3, 4\})$
Question: Does there exists a colouring $\phi : V \to \{1, 2, 3, 4\}$, for which $\phi(v) \in \mathcal{L}(v)$, $\phi^{-1}(4)$ is a clique, and for any edge $uw \in E$ the set $\phi^{-1}(\{u, w\})$ is different from $\{1\}$, $\{2\}$ and $\{1, 3\}$?

It is known that a polynomial algorithm for the 3-COMPATIBLE COLOURING problem implies a polynomial algorithm for the STUBBORN PROBLEM (as stated in [8]). Due to their role as the the last unresolved case in the classification of Cameron et al., the problems attracted quite a lot of attention. In particular, the polynomial status of 3-COMPATIBLE COLOURING or STUBBORN PROBLEM was mentioned as an open problem in numerous places including [1, 3, 5, 6, 7, 8, 10, 11, 12, 15].

Our results In this paper we present a polynomial time algorithm for the 3-COMPATIBLE COLOURING problem and hence for the STUBBORN PROBLEM, resolving a long standing open problem in the full-CSP dichotomy project:

Theorem 3. *There exists a $O(|(V, \mathcal{C})|^{3.5})$ algorithm for the 3-*COMPATIBLE COLOURING *problem, where $|(V, \mathcal{C})| = O(|V|^2)$ is the size of the instance.*

Theorem 4. *There exists a $O(|G|^7)$ algorithm for the* STUBBORN PROBLEM*, where $|G| = O(|V| + |E|)$ is the size of the instance.*

Our results prove the dichotomy for the k-COMPATIBLE COLOURING problem. Moreover, combining with results by Cameron at al. [3] we finish the matrix classification up to size 4×4 for the LIST MATRIX PARTITION problem proving that quasi-dichotomy can be strengthened to the classical dichotomy and hence improve results of Feder et al. [9].

Theorem 5. *Let M be a symmetrical $r \times r$ matrix whose entries are subsets of $\{0, 1\}$. If $r \leq 4$ then for M the* LIST MATRIX PARTITION *problem is either NP-complete or solvable in polynomial time.*

In the literature one can also find a list version of the 3-COMPATIBLE COLOURING problem, where each vertex v is additionally equipped with a set $S_v \subseteq \{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$; and we demand that the colouring we construct satisfies additionally $\phi(v) \in S_v$. It is known that the list version of the 3-COMPATIBLE COLOURING

problem can be reduced to the original 3-COMPATIBLE COLOURING problem for instance using gadgets described in Appendix A.

The 3-COMPATIBLE COLOURING problem came to our attention when posted by Marx in the open problems list from Dagstuhl Seminar 09511 on *Parameterized complexity and approximation algorithms* [6]. Marx suspected that Fixed Parameter Tractability tools and intuitions may be useful either to design a polynomial time algorithm or a quasi-polynomial lower bound. While the final version of the algorithm is elementary and uses no tools from the parametrized complexity setting, our reasoning was heavily influenced by a technique called *iterative compression*, developed by Reed et al. [17].

Outline of the paper In Section 2 we investigate the structure of solutions for the 2-COMPATIBLE COLOURING problem (i.e., finding a split graph structure). In Section 3 we present our algorithm where Section 4 is devoted to its correctness and Section 5 to its time complexity. The correctness of our algorithm is not hard, hence an advanced reader may skip this section. However, the proof of the time complexity of our algorithm is not trivial and relies on interesting combinatorial facts included in Lemma 17.

We were unable to find a reduction from the STUBBORN PROBLEM to the 3-COMPATIBLE COLOURING problem in literature. Hence for the sake of completeness, we present our own reduction in Appendix A.

Notation We assume that we are given an input to the 3-COMPATIBLE COLOURING problem: an undirected complete graph $G = (V, E)$ with a colouring of edges $\mathcal{C} : E \to \{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$ (we denote the colours by $\mathcal{R}$, $\mathcal{G}$ and $\mathcal{B}$). For a subset of vertices $X \subseteq V$ by $G[X]$ we denote the subgraph induced by X. For a subset of edges $E' \subseteq E$ by $V(E')$ we denote the set of all endpoints of edges in E'. Similarly, for a subset of vertices $V' \subseteq V$ by $E(V')$ we denote the set of edges with both endpoints in the set V'.

2 Colouring with two colours — preliminaries

We first consider the structure of 2-COMPATIBLE COLOURING. Let W be such a set of vertices that $\mathcal{C}$ restricted to $E(W)$ has only two values, say $\mathcal{R}$ and $\mathcal{B}$. We look for all feasible colourings $\phi : W \to \{\mathcal{R}, \mathcal{B}\}$.

Definition 6. We say a vertex $v \in W$ is *interesting* if there exist two feasible colourings ϕ_1, ϕ_2 of W into $\mathcal{R}$ and $\mathcal{B}$ such that $\phi_1(v) = \mathcal{R}$ and $\phi_2(v) = \mathcal{B}$. Otherwise a vertex is *boring*.

In particular, if there is no feasible colouring of W, all vertices of W are boring.

Lemma 7. *Let u, v, w be three such vertices in W that the $\mathcal{C}(uv) = \mathcal{C}(vw) \neq \mathcal{C}(uw)$. Then v is boring, as it does not admit a feasible colouring with $\phi(v) = \mathcal{C}(vw)$.*

Proof. Assume without loss of generality that $\mathcal{C}(uv) = \mathcal{C}(vw) = \mathcal{R}$ and $\mathcal{C}(uw) = \mathcal{B}$. Assume there is a feasible colouring ϕ of $\{u, v, w\}$ in which $\phi(v) = \mathcal{R}$. Then we would have to have $\phi(u) = \phi(w) = \mathcal{B}$ (as $\mathcal{C}(uv) = \mathcal{C}(vw) = \mathcal{R}$), but this contradicts $\mathcal{C}(uw) = \mathcal{B}$. As any feasible colouring of W restricted to $\{u, v, w\}$ is a feasible colouring of $\{u, v, w\}$, v cannot be interesting. □

Lemma 8. *Let $I \subseteq W$ be the set of interesting vertices in W. Then $\mathcal{C}$ restricted to $E(I)$ has only one value (that is all the edges in $E(I)$ are of a single colour).*

Moreover, there exists an algorithm which either finds a boring vertex and the colour it cannot have, or returns NO if all vertices are interesting. The algorithm works in $O(|W|^2)$ time.

Proof. If all the edges of $E(W)$ are of the same colour (without losing generality $\mathcal{R}$), every vertex $v \in W$ is interesting, as when one sets $\phi(w) = \mathcal{B}$ for $w \neq v$, then any value of $\phi(v)$ makes ϕ a feasible colouring. Therefore, in this case the answer of the algorithm is „NO". This check can be performed in $O(|W|^2)$ time.

Now assume we found two edges u_1v_1 and u_2v_2 of different colours. If these edges share an endpoint, e.g. $v_1 = v_2$, then there is a multicoloured triangle (u_1, u_2, v_1). On the other hand if all the endpoints are different, then the edge u_1u_2 has a different colour from one of the edges u_1v_1, u_2v_2. Therefore, one of the triples (u_1, u_2, v_1) or (u_1, u_2, v_2) forms a multicoloured triangle. In each case the multicoloured triangle gives us a boring vertex with its inadmissible colour as in Lemma 7 in constant time. □

3 The algorithm

3.1 Outline of the algorithm

Let $v_1, v_2, \ldots, v_n$ to be an arbitrary order on V. Suppose we have an instance $(V, \mathcal{C})$ of the 3-COMPATIBLE COLOURING problem. Let $V_i = \{v_1, v_2, \ldots, v_i\}$, and let $\mathcal{C}_i$ be the restriction of $\mathcal{C}$ to edges in $E(V_i)$. Notice that if ϕ is a solution for $(V_i, \mathcal{C}_i)$, then ϕ restricted to V_j is a solution to $(V_j, \mathcal{C}_j)$ for any $j < i$. Thus, in particular, if there is a positive answer to $(V, \mathcal{C})$, then there is a positive answer to any $(V_i, \mathcal{C}_i)$.

We proceed by building a solution for each $(V_i, \mathcal{C}_i)$. Obviously we may start the induction with an empty set V_0 and empty function $\mathcal{C}_0$. If for some i we show there is no solution, we return NO as an answer to the original $(V, \mathcal{C})$ instance. Moreover, when building the solution to $(V_i, \mathcal{C}_i)$ we assume we are given some solution to $(V_{i-1}, \mathcal{C}_{i-1})$. Thus, we can focus on a situation in which we solve an instance $(V, \mathcal{C})$ and we already have a feasible colouring ϕ_0 for $(V \setminus \{v_0\}, \mathcal{C})$ for one fixed vertex v_0. We use this feasible colouring ϕ_0 to deeply exploit the colouring of the graph $G[V \setminus \{v_0\}]$ which is a crucial part in designing our algorithm. This type of reasoning is one of the key parts of the aforementioned *iterative compression* technique used in the Fixed Parameter Tractability community.

In each step of the algorithm we have a division of V into eighteen sets, six corresponding to each of the three colours. The algorithm is a branching algorithm — we perform operations which either simply move the vertices around, or branch out into several instances. Then we resolve each branch recursively, and if we find a feasible colouring in any of them, we return this colouring, while if all the branches return NO, we return NO. We follow a naming convention in which if $\mathcal{X}$ is one of the colours in $\{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$, then $\mathcal{Y}$ and $\mathcal{Z}$ are the other two.

Consider any colour $\mathcal{X} \in \{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$. The sets corresponding to this colour are $\texttt{Free}^{\mathcal{X}}$, $\texttt{ToDo}^{\mathcal{X}}$, $\texttt{Set}^{\mathcal{X}}$, $\texttt{ToSet}^{\mathcal{X}}$, $\texttt{Not}\mathcal{Y}^{\mathcal{X}}$ and $\texttt{Not}\mathcal{Z}^{\mathcal{X}}$. The intuitive meanings of these sets are as follows:

- $\texttt{Free}^{\mathcal{X}}$ — the "free" vertices of colour $\mathcal{X}$ — those, which were of colour $\mathcal{X}$ in ϕ_0 and our algorithm has not yet gained any information about them;
- $\texttt{ToDo}^{\mathcal{X}}$ — the "to do" vertices of colour $\mathcal{X}$ — those, which were of colour $\mathcal{X}$ in ϕ_0, but our algorithm already learned they will not be of colour $\mathcal{X}$ in the new colouring;
- $\texttt{Set}^{\mathcal{X}}$ — the "set" vertices of colour $\mathcal{X}$ — those which our algorithm has already determined to be of colour $\mathcal{X}$;
- $\texttt{ToSet}^{\mathcal{X}}$ — the "to set" vertices of colour $\mathcal{X}$ — those which are determined to be of colour $\mathcal{X}$ in the new colouring, but we have to update the current division of V before we put them into $\texttt{Set}^{\mathcal{X}}$;
- $\texttt{Not}\mathcal{Y}^{\mathcal{X}}$ and $\texttt{Not}\mathcal{Z}^{\mathcal{X}}$ — the "not $\mathcal{Y}$" and "not $\mathcal{Z}$" vertices of colour $\mathcal{X}$ — those which were of colour $\mathcal{X}$ in ϕ_0, and we already know they will not be of colour $\mathcal{Y}$ (or $\mathcal{Z}$, respectively) in the new colouring.

This information can be represented by associating with each vertex the colour assigned to it by ϕ_0 and the subset $S(v) \subseteq \{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$ of colours which are still admissible as values of $\phi(v)$. Such an approach would certainly streamline any implementation of the algorithm, but we think that naming each set separately helps underline the role each particular set plays in the analysis — thus the choice of this method of presentation.

To start the algorithm we put $\phi_0^{-1}(\mathcal{R})$ into $\texttt{Free}^{\mathcal{R}}$, $\phi_0^{-1}(\mathcal{B})$ into $\texttt{Free}^{\mathcal{B}}$ and $\phi_0^{-1}(\mathcal{G})$ into $\texttt{Free}^{\mathcal{G}}$. There are three possible colours we can give to v_0, thus we branch out into three cases, putting v_0 into $\texttt{ToSet}^{\mathcal{B}}$, $\texttt{ToSet}^{\mathcal{R}}$ or $\texttt{ToSet}^{\mathcal{G}}$.

The algorithm uses two subprocedures — shifting a vertex (from $\texttt{ToSet}^{\mathcal{X}}$ to $\texttt{Set}^{\mathcal{X}}$) and resolving a set $\texttt{ToDo}^{\mathcal{X}}$. As long as any of the sets $\texttt{ToSet}^{\mathcal{X}}$ is non–empty, we shift vertices from this set. If all sets $\texttt{ToSet}^{\mathcal{X}}$ are empty, but there is a non–empty set $\texttt{ToDo}^{\mathcal{X}}$, we resolve the set $\texttt{ToDo}^{\mathcal{X}}$. If all the sets $\texttt{ToSet}^{\mathcal{X}}$ and $\texttt{ToDo}^{\mathcal{X}}$ are empty, we claim that setting $\phi(v) = \mathcal{X}$ for $v \in \texttt{Set}^{\mathcal{X}} \cup \texttt{Free}^{\mathcal{X}} \cup \texttt{Not}\mathcal{Y}^{\mathcal{X}} \cup \texttt{Not}\mathcal{Z}^{\mathcal{X}}$ is a feasible solution and return it.

3.2 Shifting a vertex

The meaning of this step is that we have a vertex v for which we have just determined that $\phi(v) = \mathcal{X}$. This gives us some information about the vertices w with $\mathcal{C}(vw) = \mathcal{X}$, which we represent by moving vertices between appropriate sets. After including the gained information in our structure we can safely move v into $\texttt{Set}^{\mathcal{X}}$.

Let $v \in \texttt{ToSet}^{\mathcal{X}}$. The procedure of shifting a vertex works as follows: we move v from $\texttt{ToSet}^{\mathcal{X}}$ to $\texttt{Set}^{\mathcal{X}}$, and then consider all $w \in V$ such that $\mathcal{C}(vw) = \mathcal{X}$. For each such vertex w we perform the appropriate action (in parentheses we give the intuitive meanings of the actions). As before, $\mathcal{Y}$ denotes any colour different than $\mathcal{X}$ and $\mathcal{Z}$ denotes the third colour different than $\mathcal{X}$ and $\mathcal{Y}$.

- If $w \in \texttt{Set}^{\mathcal{X}}$ return NO from this branch (we have two vertices for which $\phi(v) = \phi(w) = \mathcal{X}$ connected with an $\mathcal{X}$–edge);
- If $w \in \texttt{Free}^{\mathcal{X}}$ move w to $\texttt{ToDo}^{\mathcal{X}}$ (w cannot be of colour $\mathcal{X}$);
- If $w \in \texttt{Not}\mathcal{Y}^{\mathcal{X}}$ move w to $\texttt{ToSet}^{\mathcal{Z}}$, where $\mathcal{Z}$ is the third colour, that is $\{\mathcal{X}, \mathcal{Y}, \mathcal{Z}\} = \{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$ (it is not of colour $\mathcal{X}$ nor $\mathcal{Y}$, thus it is of colour $\mathcal{Z}$);
- If $w \in \texttt{ToDo}^{\mathcal{Y}}$ move w to $\texttt{ToSet}^{\mathcal{Z}}$, where $\mathcal{Z}$ is as above (again, w is neither of colour $\mathcal{Y}$ nor $\mathcal{X}$, so it is of colour $\mathcal{Z}$);
- If $w \in \texttt{Free}^{\mathcal{Y}}$ move w to $\texttt{Not}\mathcal{X}^{\mathcal{Y}}$ (w cannot be of colour $\mathcal{X}$);
- If $w \in \texttt{Not}\mathcal{Z}^{\mathcal{Y}}$ move w to $\texttt{ToSet}^{\mathcal{Y}}$ (w cannot be of colour $\mathcal{Z}$ nor $\mathcal{X}$);
- If $w \in \texttt{ToDo}^{\mathcal{X}}$, $w \in \texttt{ToSet}^{\mathcal{X}}$, $w \in \texttt{Not}\mathcal{X}^{\mathcal{Y}}$, $w \in \texttt{ToSet}^{\mathcal{Y}}$ or $w \in \texttt{Set}^{\mathcal{Y}}$, do nothing.

3.3 Resolving a set

Consider a non–empty set $\texttt{ToDo}^{\mathcal{X}}$. The meaning of this step is that we have a set of vertices that were of colour $\mathcal{X}$ in ϕ_0, but we see they cannot be of colour $\mathcal{X}$ in ϕ. Thus, there are no $\mathcal{X}$–edges in $E(\texttt{ToDo}^{\mathcal{X}})$, and we have to colour $\texttt{ToDo}^{\mathcal{X}}$ into the two remaining colours. If there are any boring vertices in $\texttt{ToDo}^{\mathcal{X}}$, we know how to colour them, so we move them to appropriate ToSet sets and go back to shifting vertices. If all vertices in $\texttt{ToDo}^{\mathcal{X}}$ are interesting, we find all possible colourings of $\texttt{ToDo}^{\mathcal{X}}$ and branch out.

We prove formally that $E(\texttt{ToDo}^{\mathcal{X}})$ contains no edges of colour $\mathcal{X}$ in Section 4. Apply the algorithm from Lemma 8 to $\texttt{ToDo}^{\mathcal{X}}$. If we find any boring vertex v which does not admit colour $\mathcal{Y}$, we move it to $\texttt{ToSet}^{\mathcal{Z}}$ and finish the resolving step. If all vertices in $\texttt{ToDo}^{\mathcal{X}}$ are interesting, we branch out into $|\texttt{ToDo}^{\mathcal{X}}|+1$ cases. We know that all the edges of $E(\texttt{ToDo}^{\mathcal{X}})$ are of one colour by Lemma 8. We check a single edge to find out which colour it is, without loss of generality assume it is $\mathcal{Y}$. If $|\texttt{ToDo}^{\mathcal{X}}|=1$ and such an edge does not exist, it does not matter which colour different than $\mathcal{X}$ we choose. In one branch we move the whole set $\texttt{ToDo}^{\mathcal{X}}$ to $\texttt{ToSet}^{\mathcal{Z}}$. In the other $|\texttt{ToDo}^{\mathcal{X}}|$ branches we choose one vertex $v \in \texttt{ToDo}^{\mathcal{X}}$, a different one in each branch, and move this vertex to $\texttt{ToSet}^{\mathcal{Y}}$ and all the other vertices to $\texttt{ToSet}^{\mathcal{Z}}$. Note that these branches correspond to all feasible colourings of $\texttt{ToDo}^{\mathcal{X}}$ using colours different than $\mathcal{X}$. Then we solve each branch recursively, if any of them returns a feasible colouring, we return it, while if all of them return NO, we return NO.

4 Correctness of the algorithm

We formally prove the correctness of the algorithm given in Section 3. A reader accustomed to such algorithms may probably only glance over this section and fill in the necessary details by him- or herself.

Formally, we do not yet know that the algorithm always terminates. In order to clarify the proof, we now assume that this indeed holds. In Section 5 we justify this assumption by showing even polynomial bounds on the algorithm's working time.

Definition 9. We say a division of V into the eighteen sets satisfies *proper invariants* if

1. For each colour $\mathcal{X}$ and for any $e \in E(\texttt{Set}^{\mathcal{X}} \cup \texttt{Free}^{\mathcal{X}} \cup \texttt{Not}\mathcal{Y}^{\mathcal{X}} \cup \texttt{Not}\mathcal{Z}^{\mathcal{X}})$ we have $\mathcal{C}(e) \neq \mathcal{X}$;
2. For each colour $\mathcal{X}$ and for any $e \in E(\texttt{ToDo}^{\mathcal{X}} \cup \texttt{Free}^{\mathcal{X}} \cup \texttt{Not}\mathcal{Y}^{\mathcal{X}} \cup \texttt{Not}\mathcal{Z}^{\mathcal{X}})$ we have $\mathcal{C}(e) \neq \mathcal{X}$;

Definition 10. A colouring $\phi : V \to \{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$ is said to be *proper* with respect to a division of V into the eighteen sets if for every colour $\mathcal{X}$ it satisfies

- $\phi(v) \neq \mathcal{X}$ for $v \in \texttt{Not}\mathcal{X}^{\mathcal{Y}}, \texttt{Not}\mathcal{X}^{\mathcal{Z}}, \texttt{ToDo}^{\mathcal{X}}$;
- $\phi(v) = \mathcal{X}$ for $v \in \texttt{ToSet}^{\mathcal{X}}$, $v \in \texttt{Set}^{\mathcal{X}}$;

We prove that the division at each step of our algorithm satisfies proper invariants. Moreover, we prove that if there exists a proper solution ϕ, then our algorithm does not return NO.

4.1 Proper invariants

Note that as ϕ_0 was a feasible colouring for $V \setminus \{v_0\}$, the proper invariants are satisfied at the start of the algorithm.

We have to check that the operations of shifting a vertex and resolving a set do not spoil proper invariants.

Firstly, we consider shifting a vertex. Assume we shift a vertex v from $\texttt{ToSet}^{\mathcal{X}}$ to $\texttt{Set}^{\mathcal{X}}$. Begin by considering the moves of vertices w with $\mathcal{C}(vw) = \mathcal{X}$. The moves $\texttt{Not}\mathcal{Y}^{\mathcal{X}} \to \texttt{ToSet}^{\mathcal{Z}}$, $\texttt{ToDo}^{\mathcal{Y}} \to \texttt{ToSet}^{\mathcal{Z}}$ and $\texttt{Not}\mathcal{Z}^{\mathcal{Y}} \to \texttt{ToSet}^{\mathcal{Y}}$ cannot spoil proper invariants since the sets $\texttt{ToSet}^{\mathcal{X}}$ are not involved in the invariants. Returning NO obviously does not spoil proper invariants. The move $\texttt{Free}^{\mathcal{X}} \to \texttt{ToDo}^{\mathcal{X}}$ decreases the number of constraints in the invariants, and $\texttt{Free}^{\mathcal{Y}} \to \texttt{Not}\mathcal{X}^{\mathcal{Y}}$ does not change the invariants.

As far as the move of v from $\texttt{ToSet}^{\mathcal{X}}$ to $\texttt{Set}^{\mathcal{X}}$ is concerned, if there were any vertices $w \in \texttt{Set}^{\mathcal{X}} \cup \texttt{Not}\mathcal{Y}^{\mathcal{X}} \cup \texttt{Not}\mathcal{Z}^{\mathcal{X}} \cup \texttt{Free}^{\mathcal{X}}$ such that $\mathcal{C}(vw) = \mathcal{X}$, the shifting algorithm removes them from the set (or returns NO for $w \in \texttt{Set}^{\mathcal{X}}$).

Thus after shifting a single vertex proper invariants still hold.

Resolving a set involves only moving vertices to the $\texttt{ToSet}$ sets, which are not constrained in the invariants, so it does not spoil the invariants as well.

4.2 Existence of a solution

Assume that at a given stage of the algorithm there is a proper colouring ϕ, which is a feasible solution to $(V, \mathcal{C})$. We prove that after performing a single step ϕ is still proper in at least one branch.

First consider shifting a vertex v from $\texttt{ToSet}^{\mathcal{X}}$ to $\texttt{Set}^{\mathcal{X}}$. As v was in $\texttt{ToSet}^{\mathcal{X}}$ and ϕ is proper, $\phi(v) = \mathcal{X}$. Thus after moving v from $\texttt{ToSet}^{\mathcal{X}}$ to $\texttt{Set}^{\mathcal{X}}$ the solution ϕ is still proper. Consider any vertex w with $\mathcal{C}(vw) = \mathcal{X}$. Then $\phi(w) \neq \mathcal{X}$. If $w \in \texttt{Set}^{\mathcal{X}}$ we have a contradiction as ϕ being a proper solution implies $\phi(w) = \mathcal{X}$. If w is moved to $\texttt{ToDo}^{\mathcal{X}}$ or $\texttt{Not}\mathcal{X}^{\mathcal{Y}}$ (from $\texttt{Free}^{\mathcal{X}}$ or $\texttt{Free}^{\mathcal{Y}}$, respectively), ϕ is still a proper solution, for the only new constraint is that $\phi(w) \neq \mathcal{X}$, which we know to be satisfied. If w was in $\texttt{Not}\mathcal{Y}^{\mathcal{X}}$, $\texttt{ToDo}^{\mathcal{Y}}$ or $\texttt{Not}\mathcal{Y}^{\mathcal{Z}}$, then $\phi(w) \neq \mathcal{Y}$ as ϕ was proper. As we additionally know that $\phi(w) \neq \mathcal{X}$, this implies $\phi(w) = \mathcal{Z}$, thus after moving w to $\texttt{ToSet}^{\mathcal{Z}}$ the solution ϕ remains proper. Thus ϕ is still proper after shifting a vertex.

Now consider resolving a set $\texttt{ToDo}^{\mathcal{X}}$. As ϕ is proper, $\phi(v) \neq \mathcal{X}$ for any $v \in \texttt{ToDo}^{\mathcal{X}}$. On the other hand, the proper invariants guarantee that $\mathcal{C}(e) \neq \mathcal{X}$ for $e \in E(\texttt{ToDo}^{\mathcal{X}})$. Thus the application of Lemma 8 is justified. If there exists a boring $v \in \texttt{ToDo}^{\mathcal{X}}$, which — according to the algorithm from Lemma 8 — cannot have $\phi(v) = \mathcal{Y}$ for any feasible colouring, we have $\phi(v) = \mathcal{Z}$. Thus after moving v to $\texttt{ToSet}^{\mathcal{Z}}$ the solution ϕ remains proper.

If all vertices are interesting, then by Lemma 8 all the edges in $E(\texttt{ToDo}^{\mathcal{X}})$ are of a single colour, say $\mathcal{Y}$, thus at most one vertex $v \in \texttt{ToDo}^{\mathcal{X}}$ satisfies $\phi(v) = \mathcal{Y}$. If there exists such a vertex, ϕ is a proper colouring for the branch in which we move v to $\texttt{ToSet}^{\mathcal{Y}}$ and all the other vertices from $\texttt{ToDo}^{\mathcal{X}}$ to $\texttt{ToSet}^{\mathcal{Z}}$. If no such vertex exists, ϕ is a proper colouring for the branch in which we move all vertices to $\texttt{ToSet}^{\mathcal{Z}}$.

Now assume that there exists any solution ϕ for the original problem $(V, \mathcal{C})$. Let $\mathcal{X} = \phi(v_0)$. Then ϕ is proper in the starting branch in which we set $v_0 \in \texttt{ToSet}^{\mathcal{X}}$ — we have $\phi(v_0) = \mathcal{X}$, and all the other vertices are in the sets $\texttt{Free}$, so we assume nothing about them.

So, finally — if there exists a solution for the original problem, our algorithm returns a solution. On the other hand, if our algorithm returns a solution, sets $\texttt{ToDo}^{\mathcal{X}}$ and $\texttt{ToSet}^{\mathcal{X}}$ are empty and the first proper invariant guarantees that it is a feasible solution to the original problem.

This allows us to formulate the following theorem:

Theorem 11. *Consider an instance $(V, \mathcal{C})$ of compatible colouring, and assume we are given a feasible colouring ϕ for $(V \setminus \{v\}, \mathcal{C})$. Then if there exists any feasible colouring for $(V, \mathcal{C})$, the algorithm described in Section 3 returns a colouring, and conversely any colouring returned by the algorithm is a feasible one for $(V, \mathcal{C})$.*

5 Time complexity bounds

Let us denote $|V|$ by n. Consider a tree of recursion for our algorithm. We actually consider three recursion trees, one for each possible choice of the set $\texttt{ToDo}^{\mathcal{X}}$ to put v_0 into.

Definition 12. By a *state* S of the algorithm we mean a division of the set V into the eighteen sets postulated by the algorithm. We denote these 18 sets by $\texttt{Free}^{\mathcal{X}}(S)$, $\texttt{ToDo}^{\mathcal{X}}(S)$, and so on, omitting the argument when it is clear what state we are considering.

By an *inner node* of the recursion tree we mean the state of the algorithm at a moment just before branching out when resolving a set $\texttt{ToDo}^{\mathcal{X}}$ with no boring vertices.

By a *leaf node* of the recursion tree we mean the state of the algorithm when it terminates a branch — either answering NO due to a failed shift operation or returning a solution due to the sets $\texttt{ToDo}^{\mathcal{X}}$ and $\texttt{ToSet}^{\mathcal{X}}$ all being empty. For the sake of analysis it is better to assume that when answering NO we first shift all vertices out the $\texttt{ToSet}^{\mathcal{X}}$ sets, disregarding conflicts, and move the vertices required by the shift. Thus we answer NO in the state when all sets $\texttt{ToSet}^{\mathcal{X}}$ are empty.

By the *descendants* of an inner node N we mean nodes that occur in any of the branches of resolving $\texttt{ToDo}^{\mathcal{X}}$ in N. This obviously gives rise to a tree structure in each of the three recursion trees, so we use the standard terms "child", "father", "root" and so on.

Each branching out takes $O(n^2)$ time for the application of Lemma 8 (not counting the time needed to solve the branches), and in total $O(n^2)$ time to prepare the branches. Between an inner node and its child a number of operations are performed, each being either shifting a single vertex (which takes $O(n)$ time) or resolving a set containing boring vertices (which takes $O(n^2)$ time).

5.1 Length of branches

Definition 13. The *potential* of a given state S of the algorithm is equal to

$$\sum_{\mathcal{X}\in\{\mathcal{R},\mathcal{G},\mathcal{B}\}} 3|\texttt{Free}^{\mathcal{X}}(S)| + 2|\texttt{ToDo}^{\mathcal{X}}(S)| + 2|\texttt{Not}\mathcal{Y}^{\mathcal{X}}(S)| + 2|\texttt{Not}\mathcal{Z}^{\mathcal{X}}(S)| + |\texttt{ToSet}^{\mathcal{X}}(S)|.$$

Lemma 14. *Shifting a single vertex and resolving a set containing a boring vertex decreases the potential.*

Proof. The move $\texttt{ToSet}^{\mathcal{X}}\rightarrow\texttt{Set}^{\mathcal{X}}$, which happens every time we shift a vertex, decreases the potential by 1. The same holds for the move $\texttt{ToDo}^{\mathcal{X}}\rightarrow\texttt{ToSet}^{\mathcal{Y}}$, which happens every time we resolve a set with a boring vertex. All the other moves associated shifting a vertex ($\texttt{Free}^{\mathcal{X}}\rightarrow\texttt{ToDo}^{\mathcal{X}}$, $\texttt{Not}\mathcal{Y}^{\mathcal{X}}\rightarrow\texttt{ToSet}^{\mathcal{Z}}$, $\texttt{ToDo}^{\mathcal{Y}}\rightarrow\texttt{ToSet}^{\mathcal{Z}}$, $\texttt{Free}^{\mathcal{Y}}\rightarrow\texttt{Not}\mathcal{X}^{\mathcal{Y}}$ and $\texttt{Not}\mathcal{Z}^{\mathcal{Y}}\rightarrow\texttt{ToSet}^{\mathcal{Y}}$) do not increase the potential. □

Lemma 15. *When we branch out while resolving a set without boring vertices, the potential in each of the branches is smaller than the potential in the original state.*

Proof. We resolve only non–empty sets. We move all vertices from $\texttt{ToDo}^{\mathcal{X}}$ to $\texttt{ToSet}^{\mathcal{Y}}$ or $\texttt{ToSet}^{\mathcal{Z}}$, each such move decreases the potential by one. □

The starting potential is $O(n)$, and decreases with each operation. Thus we have the following corollary:

Corollary 16. *We perform $O(n)$ operations (i.e., shifts, resolves of boring vertices or branches) on each path from a starting node to any leaf of the recursion tree.*

5.2 Number of leaves

We begin by formulating the lemma which is crucial to estimating the number of leaves:

Lemma 17. *Consider any inner node S of the recursion tree formed immediately before resolving a set $\texttt{ToDo}^{\mathcal{X}}$ containing no boring vertices. Let $S_0,\ldots,S_k$ be the children of S. Let $U_i^{\mathcal{X}} = \texttt{Free}^{\mathcal{X}}(S_i) \cup \texttt{ToDo}^{\mathcal{X}}(S_i)$. Then the sets $U_i^{\mathcal{X}}$ are disjoint subsets of the set $\texttt{Free}^{\mathcal{X}}(S)$.*

Proof. The sets $U_i^{\mathcal{X}}$ are subsets of $\texttt{Free}^{\mathcal{X}}(S)$ by the definition of resolving a set. By application of Lemma 8 we conclude that $E(\texttt{ToDo}^{\mathcal{X}}(S))$ contains edges of a single colour (different than $\mathcal{X}$ due to the proper invariants), say $\mathcal{Y}$ (if $\texttt{ToDo}^{\mathcal{X}}(S)$ consists of a single vertex, take as $\mathcal{Y}$ any colour different than $\mathcal{X}$). Denote the vertices of $\texttt{ToDo}^{\mathcal{X}}$ by $v_1, v_2, \ldots, v_k$. Without losing generality assume that S_0 corresponds to the branch where the whole $\texttt{ToDo}^{\mathcal{X}}$ is moved to $\texttt{ToSet}^{\mathcal{Z}}$, while S_i for $i \geq 1$ corresponds to the branch where the vertex v_i is the only one moved to $\texttt{ToSet}^{\mathcal{Y}}$. Let A_i be the set of those vertices w in $\texttt{Free}^{\mathcal{X}}(S)$ for which $\mathcal{C}(v_i w) = \mathcal{Z}$. By the second proper invariant we know that for $w \in \texttt{Free}^{\mathcal{X}}(S) \setminus A_i$ we have $\mathcal{C}(v_i w) = \mathcal{Y}$ — there are no $\mathcal{X}$–edges in $\texttt{Free}^{\mathcal{X}} \cup \texttt{ToDo}^{\mathcal{X}}$.

Consider the branch in which we move the whole set $\texttt{ToDo}^{\mathcal{X}}$ to $\texttt{ToSet}^{\mathcal{Z}}$. When we shift any vertex v_i to $\texttt{Set}^{\mathcal{Z}}$, every vertex $w \in A_i$ that was still left in $\texttt{Free}^{\mathcal{X}}$ is moved to $\texttt{Not}\mathcal{Z}^{\mathcal{X}}$. Similarly, every vertex $w \in A_i$ now contained in $\texttt{ToDo}^{\mathcal{X}}$ is moved to $\texttt{ToSet}^{\mathcal{Y}}$. Also, neither a shift nor resolving a boring vertex moves any vertex into $\texttt{Free}^{\mathcal{X}} \cup \texttt{ToDo}^{\mathcal{X}}$. Thus after all the k shifts of vertices that were in $\texttt{ToDo}^{\mathcal{X}}$ before branching, $U_0^{\mathcal{X}}$ is disjoint from $A_1 \cup A_2 \cup \ldots \cup A_k$. Similarly, in the branch where v_i is moved to $\texttt{ToSet}^{\mathcal{Y}}$ and the other v_js are moved to $\texttt{ToSet}^{\mathcal{Z}}$, after all the shifts $U_i^{\mathcal{X}}$ is disjoint from $A_1 \cup A_2 \cup \ldots \cup A_{i-1} \cup (\texttt{Free}^{\mathcal{X}} \setminus A_i) \cup A_{i+1} \cup \ldots \cup A_k$.

Consider any two branches and the associated sets $U_i^{\mathcal{X}}$. Assume the first of these branches moved the jth vertex to $\texttt{ToSet}^{\mathcal{Y}}$ (at least one of them had to move some vertex to $\texttt{ToSet}^{\mathcal{Y}}$). Then $U_j^{\mathcal{X}}$ for the first branch is contained in A_j, while $U_i^{\mathcal{X}}$ for the second is disjoint from A_j. This proves the thesis. □

We aim to prove that each recursion tree has $O(n^3)$ leaf nodes. Consider the following definition:

Definition 18. Let $U^{\mathcal{X}}(S) = \texttt{Free}^{\mathcal{X}}(S) \cup \texttt{ToDo}^{\mathcal{X}}(S)$, as above. The *mass* of a given state S of the algorithm is equal to

$$m(S) = (|U^{\mathcal{R}}(S)| + 1)(|U^{\mathcal{G}}(S)| + 1)(|U^{\mathcal{B}}(S)| + 1).$$

The mass of the root of the recursion tree is obviously $O(n^3)$, while the mass of each leaf is at least 1. As previously, shifting a vertex and resolving a boring vertex do not increase the mass of a state, as they cannot increase the sizes of sets $U^{\mathcal{R}}, U^{\mathcal{G}}, U^{\mathcal{B}}$. We will prove that for any node of the tree the mass of the node is not smaller than the sum of masses of its sons. Clearly this leads to the conclusion that the mass of the root node is greater or equal to the sum of masses of all the leaves, which, along with the bounds for the masses of the root and the leaves, shows that there are at most $O(n^3)$ leaves. Therefore, all we need is the following lemma:

Lemma 19. *Consider any node S of the recursion tree formed immediately before resolving a set $\texttt{ToDo}^{\mathcal{X}}$ containing no boring vertices. Let $S_0, S_1, \ldots, S_k$ be the children of S. Then*

$$m(S) \geq \sum_{i=0}^{k} m(S_i).$$

Proof. Without loss of generality, assume that we are resolving the set $\texttt{ToDo}^{\mathcal{R}}$ in S. Neither resolving a set $\texttt{ToDo}^{\mathcal{R}}$, shifting a vertex nor resolving a boring vertex can increase the size of sets $U^{\mathcal{G}}, U^{\mathcal{B}}$, so for all $i = 0, 1, \ldots, k$ we have that $|U^{\mathcal{G}}(S_i)| \leq |U^{\mathcal{G}}(S)|$ and $|U^{\mathcal{B}}(S_i)| \leq |U^{\mathcal{B}}(S)|$. Moreover, the number of branches (that is, the number of sons of S) is equal exactly to $k + 1 = |\texttt{ToDo}^{\mathcal{R}}(S)| + 1$ — one branch for every vertex in $\texttt{ToDo}^{\mathcal{R}}$ to be assigned the "other" colour, and one branch for all the vertices having the same colour. Thus, application of Lemma 17 immediately yields:

$$\sum_{i=0}^{k} (|U^{\mathcal{R}}(S_i)| + 1) = |\texttt{ToDo}^{\mathcal{R}}(S)| + 1 + \sum_{i=0}^{k} |U^{\mathcal{R}}(S_i)| \leq |\texttt{ToDo}^{\mathcal{R}}(S)| + 1 + |\texttt{Free}^{\mathcal{R}}(S)| = |U^{\mathcal{R}}(S)| + 1.$$

Multiplying this inequality by $(|U^{\mathcal{G}}(S)|+1)(|U^{\mathcal{B}}(S)|+1)$ we get

$$m(S) = (|U^{\mathcal{R}}(S)|+1)(|U^{\mathcal{G}}(S)|+1)(|U^{\mathcal{B}}(S)|+1) \geq \sum_{i=0}^{k}(|U^{\mathcal{R}}(S_i)|+1)(|U^{\mathcal{G}}(S)|+1)(|U^{\mathcal{B}}(S)|+1) \geq \sum_{i=0}^{k} m(S_i).$$

□

As there are $O(n)$ operations on the path to each leaf, and each operation takes $O(n^2)$ time, we have the following corollary:

Corollary 20. *The total run–time of the algorithm described in Section 3 is $O(n^6)$ for each new vertex v_0. The whole algorithm runs in $O(n^7)$ time.*

Acknowledgements

We would like to thank Daniel Marx for showing us this problem, and for a number of suggestions that helped make this paper significantly better, especially regarding Lemma 19.

References

[1] Open problem garden. `http://garden.irmacs.sfu.ca/`.

[2] Andrei A. Bulatov. A dichotomy theorem for constraints on a three-element set. In *Proc. of FOCS'02*, pages 649–658, 2002.

[3] Kathie Cameron, Elaine M. Eschen, Chính T. Hoàng, and R. Sritharan. The complexity of the list partition problem for graphs. *SIAM J. Discrete Math.*, 21(4):900–929, 2007.

[4] M. Chudnovsky, N. Robertson, P.D. Seymour, and R.Thomas. The strong perfect graph theorem. *Ann. Math.*, (164):51–229, 2006.

[5] Simone Dantas, Celina M. Herrera de Figueiredo, Sylvain Gravier, and Sulamita Klein. Finding h-partitions efficiently. *ITA*, 39(1):133–144, 2005.

[6] Eric D. Demaine, Mohammad Taghi Hajiaghayi, and Dániel Marx. Open problems from dagstuhl seminar 09511, 2009.

[7] Tomás Feder and Pavol Hell. List constraint satisfaction and list partition. manuscript. `http://theory.stanford.edu/~tomas/listpart.ps`.

[8] Tomás Feder and Pavol Hell. Full constraint satisfaction problems. *SIAM J. Comput.*, 36(1):230–246, 2006.

[9] Tomás Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. Complexity of graph partition problems. In *Proc. of STOC'99*, pages 464–472, 1999.

[10] Tomás Feder, Pavol Hell, Daniel Král, and Jiri Sgall. Two algorithms for general list matrix partitions. In *Proc. of SODA'05*, pages 870–876, 2005.

[11] Tomás Feder, Pavol Hell, David G. Schell, and Juraj Stacho. Dichotomy for tree-structured trigraph list homomorphism problems. Preprint submitted to Elsevier.

[12] Tomás Feder, Pavol Hell, and Kim Tucker-Nally. Digraph matrix partitions and trigraph homomorphisms. *Discrete Applied Mathematics*, 154(17):2458–2469, 2006.

[13] Tomás Feder and Moshe Y. Vardi. Monotone monadic SNP and constraint satisfaction. In *Proc. of STOC'93*, pages 612–622, 1993.

[14] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[15] Pavol Hell. From graph colouring to constraint satisfaction: There and back again. In *Topics in Discrete Mathematics*, pages 407–432, 2006.

[16] Pavol Hell and Jaroslav Nesetril. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, 2:143–163, 2008.

[17] Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

[18] T. J. Shaefer. The complexity of satisfiability problems. In *Proc. of STOC'78*, pages 216–226, 1978.

A Stubborn problem reduction

Recall that the *stubborn problem* [3] can be defined as follows:

STUBBORN PROBLEM
Input: An undirected graph $G = (V, E)$ and a constraint function $\mathcal{L} : V \to \mathcal{P}(\{1, 2, 3, 4\})$
Question: Does there exists a colouring $\phi : V \to \{1, 2, 3, 4\}$, for which $\phi(v) \in \mathcal{L}(v)$, $\phi^{-1}(4)$ is a clique, and for any edge $uw \in E$ the set $\phi^{-1}(\{u, w\})$ is different from $\{1\}$, $\{2\}$ and $\{1, 3\}$.

We show that this problem can be reduced to the 3-COMPATIBLE COLOURING problem.

A.1 Gadgets

We begin by showing two gadgets which can be implemented in 3-COMPATIBLE COLOURING. Consider any 3-COMPATIBLE COLOURING instance $(V, \mathcal{C})$.

Definition 21. By adding a *type one $\mathcal{X}$–gadget* to G we mean adding 4 vertices v_1, v_2, v_3, v_4 with $\mathcal{C}(v_1v_2) = \mathcal{C}(v_3v_4) = \mathcal{Y}$ and $\mathcal{C}(v_1v_3) = \mathcal{C}(v_1v_4) = \mathcal{C}(v_2v_3) = \mathcal{C}(v_2v_4) = \mathcal{Z}$, and for any v outside the gadget we have $\mathcal{C}(v_1v) = \mathcal{C}(v_2v) = \mathcal{C}(v_3v) = \mathcal{C}(v_4v)$. The exact restraints can be defined arbitrarily.

Lemma 22. *Consider an instance $(V, \mathcal{C})$ of 3-*COMPATIBLE COLOURING *and a set $S \subset V$. Let $(V', \mathcal{C}')$ be V after adding a type one $\mathcal{X}$–gadget. We put $\mathcal{C}(v_iv) = \mathcal{X}$ for $v \in S$ and $\mathcal{C}(v_iv) = \mathcal{Y}$ for $v \in V \setminus S$. Then $(V', \mathcal{C}')$ has a solution iff $(V, \mathcal{C})$ has a solution ϕ with $\phi^{-1}(\mathcal{X}) \cap S = \emptyset$.*

Proof. If $(V, \mathcal{C})$ has a solution as above, we put $\phi' = \phi$ on V, $\phi'(v_i) = \mathcal{X}$. This is trivially a solution to $(V', \mathcal{C}')$.

On the other hand, direct check shows that any feasible colouring of $\{v_1, v_2, v_3, v_4\}$ has to have at least one vertex of colour $\mathcal{X}$. Thus any feasible colouring of V' restricted to V is a restricted colouring satisfying the conditions above. □

As a corollary we deduce that adding type one gadgets enable us to implement constraint lists — in addition to the standard 3-COMPATIBLE COLOURING structure we can demand that an arbitrary set of vertices *is not* of colour $\mathcal{R}$ (or $\mathcal{G}$ or $\mathcal{B}$) by adding a type one $\mathcal{R}$–gadget and connecting it to the set by edges of colour $\mathcal{R}$. Further on we assume we added to the graph type one gadgets of all three colours.

Definition 23. Let $u, w \in G$. By adding a *type two $\mathcal{X}$–gadget* to uw we mean adding two vertices v_0, v_1 with $\mathcal{C}(uv_0) = \mathcal{C}(wv_1) = \mathcal{X}$, $\mathcal{C}(uv_1) = \mathcal{C}(wv_0) = \mathcal{Y}$, $\mathcal{C}(v_0v_1) = \mathcal{Z}$. Moreover, we assume both v_0 and v_1 are connected by $\mathcal{Y}$ edges to a type one $\mathcal{Y}$–gadget. All the other edges connecting v_0 and v_1 to the graph are also $\mathcal{Y}$ edges.

Lemma 24. *Let $(V, \mathcal{C})$ be an instance of 3-*COMPATIBLE COLOURING *and let $(V', \mathcal{C}')$ be the same instance after adding a type two $\mathcal{X}$–gadget to the edge uw. Then $(V', \mathcal{C}')$ has a feasible colouring iff $(V, \mathcal{C})$ has a feasible colouring in which at least one endpoint of uw is not of colour $\mathcal{X}$.*

Proof. Note that in any feasible colouring of $(V', \mathcal{C}')$ neither v_0 nor v_1 can be coloured $\mathcal{Y}$ due to the type one $\mathcal{Y}$–gadget. Moreover, at least one of them is not coloured $\mathcal{Z}$ due to the $\mathcal{Z}$–edge connecting them. Thus at least one of them has to be coloured $\mathcal{X}$, and — due to the $\mathcal{X}$–edges v_0u and v_1w — at least one of uw has to be of a colour different than $\mathcal{X}$. Thus the restriction of a feasible colouring on $(V', \mathcal{C}')$ to $(V, \mathcal{C})$ is a colouring as above.

On the other hand, any colouring of $(V, \mathcal{C})$ as above can be extended to a proper colouring of $(V', \mathcal{C}')$ by putting $\phi'(v_0) = \mathcal{X}$ if $\phi'(u) \neq \mathcal{X}$ and $\phi'(v_0) = \mathcal{Z}$ if $\phi'(u) = \mathcal{X}$, and the same for v_1 and w. □

This gadget allows us to add additional edge constraints to the graph (as if we were able to draw multiple edges).

A.2 Reduction

Consider any instance $((V, E), \mathcal{L})$ of the STUBBORN PROBLEM problem. We construct an equivalent instance $(V', \mathcal{C}')$ of 3-COMPATIBLE COLOURING as follows:

- If $uw \in E$, put $\mathcal{C}'(uw) = \mathcal{R}$;
- If $uw \notin E$, put $\mathcal{C}'(uw) = \mathcal{B}$;
- Add type one gadgets of all three colours to V;
- If $2 \notin \mathcal{L}(v)$, connect v to the $\mathcal{R}$–gadget by $\mathcal{R}$–edges;
- If $4 \notin \mathcal{L}(v)$, connect v to the $\mathcal{B}$–gadget by $\mathcal{B}$–edges;
- If $\{1, 3\} \cap \mathcal{L}(v) = \emptyset$, connect v to the $\mathcal{G}$–gadget by $\mathcal{G}$–edges;
- If $uw \in E$ and $3 \notin \mathcal{L}(u) \cap \mathcal{L}(w)$, add a type two $\mathcal{G}$–gadget to uw.
- All the edges connecting a type one $\mathcal{X}$–gadget to the rest of the graph not defined above are $\mathcal{Y}$–edges.

We set out to prove the following theorem:

Theorem 25. *There exists a feasible solution to the instance $((V, E), \mathcal{L})$ of* STUBBORN PROBLEM *iff there exists a feasible solution to the instance $(V', \mathcal{C}')$ of 3-*COMPATIBLE COLOURING.

Proof. If we have a solution ϕ to $((V, E), \mathcal{L})$, consider the following ϕ' for $v \in V$: if $\phi(v) = 2$, we put $\phi'(v) = \mathcal{R}$, if $\phi(v) = 4$, we put $\phi'(v) = \mathcal{B}$ and if $\phi(v) \in \{1, 3\}$ we put $\phi'(v) = \mathcal{G}$. This is a feasible solution to $(V', \mathcal{C}')$ without the added gadgets — as there are no $\mathcal{G}$ edges between vertices from V, $\phi^{-1}(4)$ is a clique so there are no $\mathcal{B}$–edges connecting two $\mathcal{B}$–vertices, and $\phi^{-1}(2)$ is an independent set, so there are no $\mathcal{R}$–edges connecting two $\mathcal{R}$–vertices. Moreover, note that as the list constraints for $((V, E), \mathcal{L})$ were satisfied, the type one gadget constraints are satisfied in $(V', \mathcal{C}')$. Finally, for any two $\mathcal{G}$–vertices we either have $uw \notin E$ or both of them were given $\phi(u) = \phi(w) = 3$. Thus list constraints for u and w allowed value 3, so there was no $\mathcal{G}$–gadget on uw. So the type two gadget constraints are satisfied as well. Therefore, using Lemmata 22 and 24 one can extend ϕ' on the whole V' obtaining a solution to $(V', \mathcal{C}')$.

In the other direction, considering a feasible solution ϕ' to $(V', \mathcal{C}')$ we can obtain a feasible solution to $((V, E), \mathcal{L})$ by putting $\phi(v) = 2$ for $\phi'(v) = \mathcal{R}$, $\phi(v) = 4$ for $\phi'(v) = \mathcal{B}$, $\phi(v) = 3$ if $\phi'(v) = \mathcal{G}$ and $3 \in \mathcal{L}(v)$ and $\phi(v) = 1$ if $\phi'(v) = \mathcal{G}$ and $3 \notin \mathcal{L}(v)$. □